Book Review—*Coding Literacy: How Programming Is Changing Writing* by Annette Vee

Antonio Byrd—University of Wisconsin

In *Coding Literacy: How Programming Is Changing Writing*, Annette Vee observes that evoking programming as a literacy is more than rhetorical flair to convince politicians and administrators that they should expand computer science curricula in public schools. She argues that when we bring to bear the theoretical concepts of literacy studies on programming, we discover that programming really is a part of literacy. Vee notes that using the tools of literacy studies leads to two new understandings about writing. First, programming becomes less a skill practiced in computer science and software development and more a *historically* and *socially* situated "species of writing" used in different ways for different purposes (5). Second, programming and the materials of writing are the foundational tools for writing, and as these tools change, so too does writing; the vast capabilities of programming augment traditional ways of reading and writing texts.

Vee draws historical, social, and theoretical parallels and comparisons between programming and literacy across four chapters to show how programming is becoming a part of literacy. In addition, she argues that the ways that programming and literacy have been linked historically and theoretically help us understand how to "account for new modes and technologies in literacy" and "the ways that computer programming is changing our practices and means of communication" (4). *Coding Literacy* offers literacy studies scholars and writing instructors a thoughtful analysis of how writing has evolved and may evolve in the future; the book is essential reading for its breadth of historical and theoretical application to computer programming that updates our notions of writing for a swiftly changing technological landscape.

In the first chapter, Vee recalls a rhetorical history of campaigns for universal reading and writing and extends that timeline into the mass programming movements of the 20th and 21st centuries. By doing so, Vee explains that programming promotion since the 1960s has used rhetoric and ideologies similar to literacy campaigns: economic productivity, individual empowerment, and creative expression. In the process of arguing for universal coding literacy, advocates reveal "which technologies and skills and ideas are now included under literacy's rubric" (46). As these early campaigns for coding literacy flourish, Vee cautions, they, like reading and writing, "reflect contemporary concerns" (91), and those concerns inadvertently (or perhaps purposely) determine who gets left behind or left out from learning programming.

Following the necessary historical overview in Chapter One, Vee continues her argument with an ontological angle on programming and writing in Chapter Two, drawing from theories of speech act, language-in-use, and *grammatization*. Vee explains that these concepts help us understand how programming works as a type of writing in clear, accessible sentences; however, readers unfamiliar with these theories may need to slow down to comprehend the parallels. In addition, Vee draws on material intelligence and sociomaterial theory to explicate the common affordances that programming shares with writing that “help us build knowledge” (105). As a “sociolingustic system,” programming both describes and performs “to different degrees” depending on audience, social context, and “the human language” (115). Programming immediately acts on explicit instructions, abstracts contextual information about those instructions, and both reduces and transforms information. With these affordances, programming can create complex software, share pieces of its procedures or all of its procedures across space and time to multiple communities, and translate information from procedure to text and vice versa. Writing and programming are not only parallel, Vee notes, but also “intersectional” (138). The kinds of communication available to writers include speech, writing, and now coding, Vee concludes. Each kind of communication allows writers to create and share different kinds of information.

Establishing programming as a type of writing creates a pathway for exploring the similar social histories of programming and writing in Chapters Three and Four. Across these chapters, Vee observes how writing evolved from a technology that government and church institutions used to manage the overflow of information to the foundation for individuals’ everyday life. This history, the author argues, can be a model for understanding how coding builds on and replaces textual writing practices as the new basis for our societal infrastructure and to see why coding may become a necessary generalizable skill. Among other theories, Vee returns to material intelligence, introduced in Chapter Two, to guide her central argument in Chapter Three: the physical materials of writing circulating in bureaucracies and businesses and then trickling down into people’s homes helped solidify writing’s essential role in organizing society. Chapter Four picks up where Chapter Three leaves off, tracking how widespread availability of writing materials led to a “literate mentality”— a shared cultural awareness that writing influences every aspect of life and is thus a necessary skill that every individual should know (Vee 182-83, 196). In both chapters, Vee shows how computation’s history overlaps with and mirrors this history of writing; the ways ubiquitous computer technology re-structures our lives, our associating our thinking and values with computation, and rising anxieties about the status of our literacy in our current technological moment direct us toward a “computational mentality” (183). Predictions of the future require some skepticism, mostly because the future never turns out the ways we expect, but Vee links historical trends to our present so well in her analysis that the chances of a computational mentality seem likely.

Scholars in rhetoric have studied programming’s social and cultural functions, but *Coding Literacy* brings programming directly to literacy scholars and offers a model for how they can extend their interests in sociocultural theories of writing into programming. Chapter Two leaves ground for further research on the sociomaterial and ontological nature of programming in multiple situations. Meanwhile, the concluding remarks in the book raise useful questions on how to teach a more diverse population of students. Based on Vee’s argument, there’s potential to bring together computer science education’s own ongoing research on teaching diverse students (Guzdial; Kross and Guo; Cooper et al.) with Writing Studies’ interests in the ideologies of literacy education and the material consequences those ideologies create. This collaboration is necessary as both fields have a unique opportunity to redirect writing’s role in continuing social inequity.

WORKS CITED

Cooper, Steve, Jeff Forbes, Armando Fox, Susanne Hambrusch, Andrew Ko, and Beth Simon. "The Importance of Computing Science Education." *Computing Community Consortium Committee of the Computing Research Association*, 2016. Web.

Guzdial, Mark. *Learner-Centered Design of Computing Education: Research on Computing for Everyone*. San Rafael: Morgan & Claypool, 2016. Print.

Kross, Sean, and Philip J. Guo. "Students, Systems, and Interactions." *ACM Conference on Learning at Scale,* London, 26-28 June 2018. ACM, 2018.

Vee, Annette. *Coding Literacy: How Computer Programming Is Changing Writing*. Cambridge: The MIT P, 2017. Print.